Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

SecureResume: A Security-Aware AI/ML Framework for Robust LLM-Based Resume Screening

Harshad Panchal

harshadpanchal122006@gmail.com

Vidyalankar Institute of Technology

Bhuban Wakode

bhuban.wakode@vit.edu.in

Vidyalankar Institution of Technology

Aaryan Khamkar

aaryan.khamkar@vit.edu.in

Vidyalankar Institution of Technology

Prof. Faiz Rangari

faiz.rangari@vit.edu.in

Vidyalankar Institution of Technology

Research Article

Keywords: Artificial Intelligence , Machine Learning, Prompt Injection, LLM Manipulation, Cybersecurity, Attackers, Hiring, Recruitment, Manipulation, Injection, Transformers, Resume, Screening

Posted Date: November 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11585909/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

SecureResume: A Security-Aware AI/ML Framework for Robust LLM-Based Resume Screening

1st Harshad Panchal
Dept. of Electronics and Computer Science
Vidyalankar Institute of Technology
Mumbai, India
harshad.panchal@vit.edu.in

2nd Bhuban Wakode
Dept. of Electronics and Computer Science
Vidyalankar Institute of Technology
Mumbai, India
bhuban.wakode@vit.edu.in

3rd Aaryan Khamkar
Dept. of Electronics and Computer Science
Vidyalankar Institute of Technology
Mumbai, India
aaryan.khamkar@vit.edu.in

4th Prof. Faiz Rangari
Project Investigator
Dept. of Electronics and Computer Science
Vidyalankar Institute of Technology
Mumbai, India
faiz.rangari@vit.edu.in

***Abstract*—Large language models (LLMs) are increasingly used for first-pass resume screening, yet the resume is supplied by the party who benefits from a favourable outcome. Applicants can therefore conceal text from human reviewers (white or microscopic fonts, zero-width characters, homoglyphs) or embed instructions that steer the model. We present SecureResume, a layered framework that (i) parses documents in a render-aware manner and removes concealed or instruction-like content, (ii) scores per-object adversarial risk with an Isolation Forest, (iii) computes Sentence-BERT job–resume similarity, and (iv) forces an LLM to emit schema-validated claims that are verified against evidence in the sanitised text, producing an evidence-adjusted ranking score. We implement the system as a FastAPI/Next.js application and evaluate its security-aware parsing layer on a controlled benchmark of 400 synthetic PDF resumes covering nine concealment techniques. After fixing two defects we found in our own parser, the rule-based layer detects all seven concealment classes it targets (280/280) with no false positives on 40 clean resumes, but misses off-page and low-contrast text; the document-level risk score, as originally formulated, saturates and is not discriminative. We report these findings candidly and outline calibration and rendering-based extensions. The LLM verification stages are specified but not yet evaluated.**

***Index Terms*—resume screening, prompt injection, large language models, document security, Sentence-BERT, anomaly detection, claim verification**

I. Introduction

Automated screening has become a standard first step in hiring, and LLM-based pipelines that summarise, grade and rank resumes promise substantial speed-ups over manual review [1]. These pipelines share a structural weakness: the document under evaluation is authored by the candidate. If any part of the extracted text reaches the model as an instruction rather than as data, the candidate can influence the verdict. This is indirect prompt injection [2], ranked first in the OWASP list of LLM risks [3].

The threat is not hypothetical. A measurement over 196,682 real resumes from a commercial platform found hidden prompt injections in roughly 1% of documents, rising over the preceding years, with more than 90% of injections consisting of hidden *data* (skills, fabricated experience) rather than explicit instructions [4]. A controlled benchmark reported attack success rates above 80% for some attack types against LLM screeners [5]. Concealment exploits the gap between what a human sees on the rendered page and what a text extractor returns.

This gap motivates three requirements for a trustworthy screener: *(R1)* extract only what a human can see and record its provenance; *(R2)* detect concealment and instruction-like content independent of its wording; and *(R3)* do not let the final score depend on unverified self-claims. SecureResume addresses these through a five-stage pipeline (Section III). The contributions of this paper are:

- A layered architecture combining security-aware parsing, ML-based anomaly scoring, semantic matching and schema-bounded LLM claim verification, with an evidence-adjusted ranking score.
- A reproducible synthetic benchmark of nine concealment techniques and an evaluation of the parsing and anomaly layers of our implementation.
- A candid analysis of failure modes, including two implementation defects and a saturation problem in the original risk score.

II. Literature Survey

Prompt injection. Perez and Ribeiro showed that appended text such as "ignore previous instructions" can redirect a model [6]. Greshake et al. generalised this to the indirect case, where the payload arrives inside content that the application

TABLE I
POSITIONING OF SECURERESUME AGAINST CLOSELY RELATED WORK.

Work	Input	Core idea	Gap addressed here
HCD/VDA [4]	PDF	visual rules + LLM/VLM	no claim verification
ResumeShield [10]	HTML/text	filters + data fence	no PDF rendering; no claim scoring
Mu et al. [5]	text	attack benchmark	no defence pipeline
SecureResume	PDF, DOCX	parse + ML + claim evidence	this work

ingests [2], and Liu et al. formalised attacks and defences in a common benchmark [7].

Attacks on resume screening. Mu et al. introduced a taxonomy over four content types (instruction, invisible keywords, fabricated experience, job manipulation) and four injection positions, reporting high success rates [5]. Zhang et al. measured the phenomenon in the wild and proposed two detectors: a Hybrid Cascade Detector (HCD), which applies rule-based visual checks (font size, colour distance, ink density) followed by LLM verification, and a Visual Discrepancy Analyzer (VDA) that compares rendered page images with extracted text using a vision-language model [4]. General-purpose injection detectors performed poorly on resumes because most injections contain no explicit instruction [4]; this is why wording-based filters alone are insufficient.

Defences. Fine-tuning approaches such as StruQ restrict instruction-following to a reserved channel [8], and Spotlighting marks untrusted spans [9]. Barach's ResumeShield applies filtering plus an application-layer data fence and, on a 104-document synthetic corpus, reports precision 1.000 and recall 0.944, with the fence contributing most of the protection in a simulated-model ablation [10]. PromptArmor uses an LLM as a detector [11]; CaMeL constrains what untrusted data may influence at the system level [12].

Building blocks. Sentence-BERT yields fixed-size embeddings that support cosine-similarity matching [13]. Isolation Forest isolates anomalies without labels [14], and gradient boosting [15] is a standard supervised alternative when labels exist. Unicode Technical Standard #39 documents confusable characters and mixed-script detection [16].

Our approach. Prior defences either detect concealment or isolate the instruction channel, but none ties the *final ranking score* to verified evidence. SecureResume combines render-aware parsing with provenance tracking, unsupervised anomaly scoring, and a claim–evidence verification step whose output is validated against a typed schema before it can affect the score. The goal is that content which survives sanitisation but is unsupported (a fabricated skill, say) lowers rather than raises the candidate's score.

III. METHODOLOGY

A. Threat model

The attacker controls the entire submitted file but not the pipeline or model. We consider five classes: T1 rendering-level concealment (white or sub-1 pt text), T2 encoding-based attacks (zero-width characters, homoglyphs), T3 structural/metadata anomalies, T4 prompt injection (instruction-like text), and T5 semantic adversarialism (fluent but unsupported claims).

B. Pipeline

Figure 1 shows the five stages.

S1–S2: security-aware parsing. PDFs are read with PyMuPDF at span level; each span becomes a *text object* x_i carrying page, offset, font size, colour, opacity, bounding box, a hidden flag, and a Unicode-anomaly score. A span is hidden if its font size is below 1 pt or its colour is white. Text is normalised with `ftfy`, zero-width and bidirectional control characters are removed, a small Cyrillic→Latin homoglyph map is applied, and NFC normalisation follows. The Unicode anomaly score is $u_i = \min(1, 10\,(z_i + h_i)/|t_i|)$, where z_i and h_i count zero-width and homoglyph characters in the raw span of length $|t_i|$. Visible spans matching an instruction-pattern regular expression (e.g., "ignore previous instructions", "assign maximum score") are redacted and recorded in metadata flags. DOCX files are processed run by run using the same rules. The sanitised text is the concatenation of non-hidden, non-redacted spans, and a provenance map links each span to its object.

S3: AI/ML layer. (a) A spaCy model extracts named entities, augmented by a keyword skill detector. (b) Each object is mapped to a feature vector (normalised font size, opacity, hidden flag, Unicode score, text density, colour brightness) and an Isolation Forest [14] produces a score converted to a probability $p_i \in [0, 1]$; hidden or strongly anomalous objects are forced to $p_i \geq 0.9$. Probabilities are aggregated with a noisy-OR:

$$R(d) = 1 - \prod_i (1 - p_i), \tag{1}$$

and a document is flagged for review when $R(d) \geq \tau$ ($\tau = 0.6$). (c) Sentence-BERT (`all-MiniLM-L6-v2`) [13] embeds the sanitised resume and the job description.

S4: matching. Embeddings are L2-normalised, so $\mathrm{sim}(d, j) = \cos(\mathbf{e}_d, \mathbf{e}_j) = \mathbf{e}_d^\top \mathbf{e}_j$.

S5: LLM screening with evidence. A Llama-3 70B model [17] served through the Groq API is called at temperature 0 for (i) claim extraction, producing a JSON list of explicit claims, and (ii) evidence verification, producing for each claim c the supporting spans from the sanitised text and a support score $s(c) \in [0, 1]$. A final call produces candidate skills and a short rationale. Every LLM output is parsed into typed Pydantic models (`ClaimAssessment`, `ScreeningResult`); malformed output triggers a safe fallback that recommends manual review rather than propagating unvalidated text. With $\bar{s}(d)$ the mean support score over claims, the evidence-adjusted score is

$$S(d, j) = \mathrm{sim}(d, j)\,\big[\alpha + (1 - \alpha)\,\bar{s}(d)\big], \quad \alpha \in [0, 1], \tag{2}$$

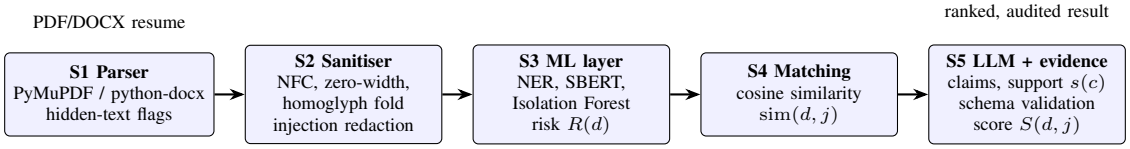


Fig. 1. SecureResume pipeline. Only sanitised text with provenance reaches the matching and LLM stages.

with $\alpha = 0.5$ by default. A document with no extractable claims receives $\bar{s} = 1$ and so is not penalised; claims with $s(c) < 0.2$ raise the T5 flag.

Implementation. The backend is FastAPI (Python 3.11) with Clerk JWT authentication; the frontend is Next.js 14 with a dashboard showing the score, flagged threats and claim table. Every stage writes to an audit log returned with the response.

IV. Results

A. Experimental setup

Because no labelled corpus of adversarial resumes exists for our setting and we did not wish to probe real applicants' documents, we built a seeded synthetic benchmark of 400 one-page PDFs generated with ReportLab: 40 clean resumes and 40 each of nine attack classes: white-text instruction (white fill), white-text keyword stuffing, 0.5 pt text, zero-width characters interleaved in keywords, Cyrillic homoglyph substitution, a visible plain-text instruction, a white paraphrased instruction that avoids the regular-expression patterns, off-page text (placed at negative x), and near-white low-contrast keywords (RGB 247). The last two are not covered by any rule in S1 and serve as honest negative controls. We ran the *actual* parser and anomaly-detection modules of our implementation; the LLM and SBERT stages were not executed because no API access was available in the evaluation environment. All numbers are therefore for stages S1–S3 only. With $n = 40$ per class, a 40/40 result has a 95% Wilson lower bound of 91.2%.

B. Defects found during evaluation

Running the code as originally written produced zero extracted text objects on all 400 documents, so nothing was flagged. The cause was calling PyMuPDF's `rawdict` mode, which returns per-character records without a span-level text field. A second defect passed PyMuPDF's 0–255 alpha value into a feature expected in $[0, 1]$, which would have dominated the Isolation Forest feature space. Both were corrected (switching to `dict` mode; dividing alpha by 255), and all results below use the corrected parser.

C. Detection of concealment

Table II and Fig. 2 give per-class results for the rule layer (hidden flag, Unicode anomaly, or injection pattern). All 280 documents in the seven targeted classes were flagged, and none of the 40 clean resumes was, giving precision 1.00 and a false-positive rate of 0/40. Recall over all 360 attack documents is 280/360 = 77.8%. The two uncovered classes were missed entirely. For the evaluation of *harm*, the more relevant quantity is whether the payload reaches the LLM: for off-page text the extractor did not return the content at all, so it never reached the sanitised text, whereas all 40 low-contrast documents passed their keywords through to the sanitised text. In the seven detected classes, the payload was removed from the sanitised text except for zero-width keywords, which were restored to their legitimate form and flagged by the Unicode score.

The paraphrased white-text class is informative: its wording evades the regular expression, yet it is caught because concealment, not wording, is the signal. This agrees with the observation that most real injections contain no explicit instruction [4] and with the "concealment dilemma" argued

TABLE II
Rule-layer detection on the synthetic benchmark ($n = 40$ per class). "Reaches LLM" = payload present in sanitised text.

Class	Detected	Reaches LLM	Signal
Clean (false positives)	0/40	–	–
White-text instruction	40/40	0/40	hidden
White-text keywords	40/40	0/40	hidden
0.5 pt text	40/40	0/40	hidden
Zero-width characters	40/40	40/40*	Unicode
Cyrillic homoglyphs	40/40	0/40	Unicode
Visible injection	40/40	0/40	pattern
White paraphrased inj.	40/40	0/40	hidden
Off-page text	0/40	0/40	none
Low-contrast keywords	0/40	40/40	none

*Keywords restored after normalisation; they were visible text.

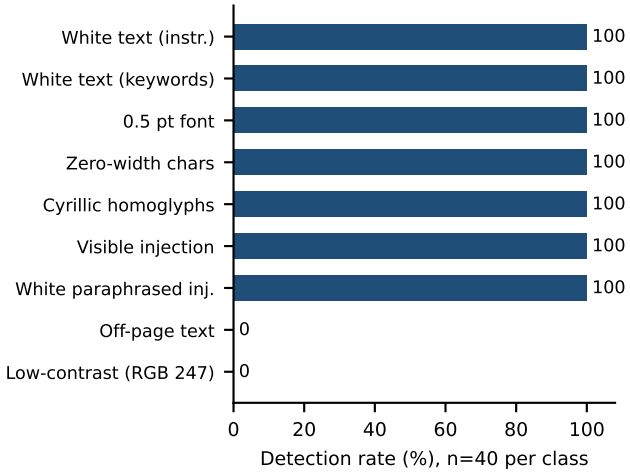


Fig. 2. Per-class detection rate of the corrected rule layer. Classes with no corresponding rule score 0.

in [10].

D. Document-level risk score

The Isolation Forest risk $R(d)$ of Eq. (1) was not discriminative. Because per-object scores are min–max normalised *within each document*, the most ordinary object is always mapped near $p = 0$ and the most atypical near $p = 1$; with about 17 objects per page the noisy-OR then saturates, and $R(d) \geq 0.999$ for every document, clean ones included. At $\tau = 0.6$ the flag therefore fires on 40/40 clean resumes. We conclude that Eq. (1) must use calibrated, document-independent probabilities, and that the deterministic rules, not $R(d)$, currently carry the detection result.

E. Cost

Measured separately on 30 clean one-page PDFs on a single CPU core of the evaluation sandbox, parsing and sanitisation took about 2.8 ms per document and Isolation Forest scoring (fitting 100 trees per document) about 85 ms. The total of roughly 88 ms is small relative to the LLM calls that follow, but the per-document model fit dominates and is another reason to replace it with a pre-trained model.

V. Conclusion

SecureResume combines render-aware parsing, anomaly scoring, semantic matching and schema-bounded claim verification so that a candidate's score depends on evidence rather than on text the candidate has hidden or merely asserted. On a controlled benchmark, the corrected parsing layer detected all seven concealment classes it was designed for without false positives, and it did so even when injected wording avoided known patterns. The evaluation also exposed real weaknesses: two implementation defects, no coverage of off-page and low-contrast text, and a risk score that saturates. The evidence-adjusted score and the LLM stages are specified and implemented but remain unevaluated. These results are on synthetic documents and should not be read as a prediction of performance on real applicant files.

VI. Future Scope

- **Calibrate $R(d)$.** Replace within-document min–max normalisation with a model trained on a corpus of benign templates, evaluating with ROC/PR curves.
- **Render-and-compare.** Rasterise pages and compare against extracted text, as in VDA [4], to catch off-page, low-contrast and layered text.
- **Live LLM evaluation.** Measure attack success as a change in score or ranking relative to the clean resume, and ablate the evidence term in Eq. (2) against a baseline without it. Add application-layer channel separation [10].
- **Fabricated visible claims (T5).** Test whether $\bar{s}(d)$ lowers scores of unsupported claims and how it affects honest candidates, including a fairness analysis across formats and templates.
- **Broader formats and human review.** Support scanned PDFs (OCR) and route flagged files to human reviewers rather than automatic rejection.

References

[1] C. Gan, Q. Zhang, and T. Mori, "Application of LLM agents in recruitment: A novel framework for automated resume screening," *J. Information Processing*, vol. 32, pp. 881–893, 2024.

[2] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. ACM Workshop AI and Security (AISec)*, 2023, pp. 79–90.

[3] OWASP Foundation, "OWASP Top 10 for LLM applications: LLM01:2025 Prompt Injection," 2025. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[4] M. Zhang, Y. Jia, Z. Tan, S. Jiang, N. Z. Gong, T. Chen, and D. Song, "Measuring real-world prompt injection attacks in LLM-based resume screening," arXiv:2605.28999, 2026 (USENIX Security 2026).

[5] H. Mu, J. Liu, K. Wan, R. Xing, X. Chen, T. Baldwin, and W. Che, "AI security beyond core domains: Resume screening as a case study of adversarial vulnerabilities in specialized LLM applications," arXiv:2512.20164, 2025.

[6] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," arXiv:2211.09527, 2022.

[7] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *Proc. USENIX Security Symp.*, 2024.

[8] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "StruQ: Defending against prompt injection with structured queries," in *Proc. USENIX Security Symp.*, 2025.

[9] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, "Defending against indirect prompt injection attacks with spotlighting," in *Conf. Applied Machine Learning for Information Security (CAMLIS)*, 2024.

[10] J. Barach, "ResumeShield: Channel separation and an open benchmark for indirect prompt injection in AI resume screening," arXiv:2609.20188, 2026.

[11] T. Shi *et al.*, "PromptArmor: Simple yet effective prompt injection defenses," arXiv:2507.15219, 2025.

[12] E. Debenedetti *et al.*, "Defeating prompt injections by design," arXiv:2503.18813, 2025.

[13] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP-IJCNLP*, 2019, pp. 3982–3992.

[14] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2008, pp. 413–422.

[15] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD*, 2016, pp. 785–794.

[16] Unicode Consortium, "Unicode Technical Standard #39: Unicode security mechanisms." [Online]. Available: https://www.unicode.org/reports/tr39/

[17] A. Dubey *et al.*, "The Llama 3 herd of models," arXiv:2407.21783, 2024.